ПОРІВНЯННЯ БАГАТОПРОЦЕСОРНОЇ ТА БАГАТОПОТОЧНОЇ РЕАЛІЗАЦІЙ ЕНТРОПІЙНОГО ПІДХОДУ ДЛЯ ІМПУТУВАННЯ ПРОПУСКІВ У ДАНИХ НА МОВІ ПРОГРАМУВАННЯ PYTHON

Олексій Земляний
ORCID: https://orcid.org/0009-0001-6157-8725
Дніпровський національний університет імені Олеся Гончара, Дніпро, Україна

Олег Байбуз
ORCID: https://orcid.org/0000-0001-7489-6952
Дніпровський національний університет імені Олеся Гончара, Дніпро, Україна

ВСТУП

У галузі науки про дані дослідницький аналіз є важливим етапом. Він допомагає вивчити характеристики даних, виявити закономірності, аномалії, очистити дані від викидів і створити початкові моделі. На цьому етапі можна визначити розподіл даних, оцінити його параметри, знайти викиди та побудувати кореляційну матрицю. Виявлення пропущених значень є серйозною проблемою попереднього аналізу, оскільки немає універсального методу для всіх випадків. Потрібно підбирати або комбінувати різні методи. Більшість моделей машинного навчання не можуть працювати з пропущеними значеннями, тому їх необхідно обробити під час підготовки даних. Одним з підходів до імпутування є ентропійний метод [1], який враховує невизначеність даних.

Ця робота присвячена порівнянню багатопроцесорної та багатопоточної реалізацій ентропійного підходу для імпутування пропусків у даних на мові програмування Python. Ми досліджуємо, як різні підходи до паралелізації можуть впливати на продуктивність імпутування. Використання багатопроцесорних і багатопотокових технологій дозволяє значно прискорити обчислювальні процеси, що є важливим для роботи з великими обсягами даних, але на мові програмування Python ця задача має свої особливості.

МЕТА І ЗАВДАННЯ

Мова програмування Python належить до сімейства інтерпретованих мов програмування, що означає, що інструкції програми виконує програма-інтерпретатор. Коли мова йде про оптимізацію обчислень, перша ідея пов’язується з багатопоточною організацією обчислень.

Але у випадку мови програмування Python з цього приводу є певні обмеження, оскільки її інтерпретатор не є повністю багатопотоковим [2]. Метою даної роботи є дослідження підходів щодо оптимізації обчислень при реалізації ентропійного підходу для імпутування пропусків у даних на мові програмування Python. В якості прикладу будемо використовувати набір даних, доступний на платформі Kaggle.com, збірний датасет UCI Heart Disease Data [3], створений кардіологічними центрами в Угорщині, Швейцарії та США.

Матеріали і методи

Інтерпретатор Python не підтримує повноцінну багатопоточність. Для роботи багатопоточних програм у Python існує глобальне блокування інтерпретатора (GIL). Даний потік повинен утримувати це блокування, щоб мати можливість безпечно працювати з об'єктами Python. Без GIL навіть прості операції можуть викликати проблеми в багатопоточних програмах: наприклад, якщо два потоки одночасно збільшують лічильник посилань на один і той самий об'єкт, лічильник може збільшитися лише один раз замість двох.

Тому встановлено правило, що тільки той потік, який отримав GIL, може працювати з об'єктами Python або викликати функції Python/C API. Щоб емулювати паралельність виконання, інтерпретатор регулярно намагається перемикати потоки. Блокування також знімається навколо потенційно блокуючих операцій вводу/виводу, таких як читання або запис файлу, щоб інші потоки Python могли працювати у цей час.

Відтак, GIL, або Global Interpreter Lock, – це м'ютекс, який обмежує доступ до інтерпретатора Python у багатопотокових середовищах, дозволяючи виконувати лише одну інструкцію одночасно. Хоча GIL забезпечує безпеку та цілісність даних, він обмежує ефективне використання багатоядерних процесорів і багатозадачність.

Python підтримує багатопоточність за допомогою модуля threading, але через GIL лише один потік може взаємодіяти з об'єктами Python у будь-який момент часу. Це ускладнює паралельне виконання завдань, особливо обчислювально інтенсивних. GIL був впроваджений, коли Python почав використовуватися для багатопотокових додатків, щоб запобігти проблемам з одночасним доступом до спільних ресурсів і забезпечити безпеку роботи з пам'яттю та об'єктами. У Python 3.2 було впроваджено систему для часткового розділення GIL, що покращило продуктивність у певних випадках. Проте GIL продовжує обмежувати багатозадачність, тому потоки в Python підходять краще для завдань, пов'язаних з очікуванням вводу-виводу,

ніж для інтенсивної обробки даних.

Взаємодія потоків з GIL може спричинити непередбачувані результати, зокрема перегони даних (race conditions), коли кілька потоків змінюють одні й ті самі дані. Один з ефективних способів обійти GIL – це спроба використати багатопроцесорну обробку (multiprocessing) замість багатопоточності. Оскільки кожен процес має власний інтерпретатор Python і GIL, вони можуть паралельно працювати на різних ядрах. Існують Python-бібліотеки, наприклад, concurrent.futures, які надають високорівневий доступ до багатопроцесорної обробки, дозволяючи легко перемикатися між пулами потоків і процесів залежно від потреб програми.

Порівняємо ефективність цих підходів з точки зору прискорення часу обчислень для задачі імпутування пропусків у даних на основі ентропійного підходу.

Результати

В нашому тестовому датасеті, як описано в роботі [4], штучно вносяться пропуски, далі проводиться імпутування пропущених значень різними реалізаціями ентропійного підходу, оцінюється середньо-квадратична похибка та час виконання алгоритмів. Розглядаються 10% пропусків, 20%, 30% та 40%. Додається патерн пропусків, з випадковою кількістю повних рядків та ознак. Виконується 10 ітерацій генерування пропусків в даних та імпутацій трьома різними реалізаціями ентропійного підходу. Обчислюється середня помилка імпутації для кожного обсягу пропусків та середній загальний час імпутування.

Імпутація даних за ентропійним підходом для кожної ознаки виконується незалежно одна від одної, тому ми можемо розпаралелити обчислення таким чином, що кожній ознаці відповідає свій потік чи процес обчислень. Ми будемо порівнювати точність імпутування та час обчислень для послідовного підходу, багатопоточного (multithreading) та багатопроцесорного (multiprocessing) підходів. В реалізаціях алгоритму імпутування будемо використовувати ThreadPoolExecutor() та ProcessPoolExecutor() з модуля concurrent.futures [5].

В нашій задачі немає проблеми перегонів даних, оскільки кожний потік чи процес вирішую свою окрему задачу та повертає результат у вигляді стовпця з імпутованими даними для ознаки, за яку він відповідає. Таким чином, нам немає потреби здійснювати блокування спільних даних в межах обчислень. Єдине наше обмеження буде стосуватись використання GIL.

Дослідження проводились на комп'ютері з процесором Intel i7-3770, що має 4 фізичних ядра та 8 логічних ядер, оскільки використовується технологія Intel® Hyper-Threading. У таблицях 1 та 2 наведено результати обчислень для двох незалежних сесій тестування. Оскільки попередні тести показали, що за швидкодією виграє реалізація з багатопроцесорним підходом, прийнято рішення також провести випробування з різною кількістю робочих процесів: 4, 8 та за замовчуванням (визначається системою).

Таблиця 1 – Порівняння ефективності різних реалізацій ентропійного методу імпутування пропусків в даних, сесія 1

Реалізація	10%	20%	30%	40%	Час виконання
Послідовний підхід	6.676	6.954	9.294	9.426	22.112261
Багатопоточний підхід	6.676	6.954	9.414	9.566	22.698414
Багатопроцесорний підхід	6.676	6.830	9.320	9.480	18.651492

Таблиця 2 – Порівняння ефективності різних реалізацій ентропійного методу імпутування пропусків в даних, сесія 2

Реалізація	10%	20%	30%	40%	Час виконання
Послідовний підхід	6.62	7.7	9.79	9.92	20.875952
Багатопоточний підхід	6.596	7.466	9.802	9.73	22.014434
Багатопроцесорний підхід, за замовчуванням	6.552	7.384	9.704	10.236	17.612596
Багатопроцесорний підхід, 4 робочі процеси	6.644	7.502	9.894	9.904	15.486620
Багатопроцесорний підхід, 8 робочих процесів	6.570	7.564	9.602	9.85	17.541755

Можна побачити, що різниці в середніх значеннях похибки в залежності від обраного підходу майже немає в тому сенсі, що кожний з підходів не може надати перевагу саме у досягненні кращої точності імпутування, оскільки оптимізація була направлена на збільшення швидкодії обчислень. Порівняння послідовного підходу та багатопоточного показує, що багатопоточний підхід не дає переваги у швидкодії, і, навіть, може її погіршити. І тільки багатопроцесорний підхід надає дійсне зменшення часу обчислень. З нашого дослідження випливає, що ми можемо покращити швидкодію, якщо кількість

робочих процесів буде дорівнювати кількості фізичних ядер процесора. Тому важливо брати до уваги саме цю характеристику.

Висновки

Ми розробили три реалізації ентропійного методу для імпутування пропусків у даних на основі послідовного, багатопоточного та багатопроцесорного підходів. В ході програмного експерименту виявлено, що багатопоточний підхід не надає переваги у порівнянні з послідовним підходом, що пояснюється наявністю обмежень у використанні GIL, описаних в документації. Можливо навіть погіршення результату, що пояснюється часом, що витрачається на перемикання між потоками. Реальне покращення результату від використання багатоядерних процесорів у випадку мови програмування Python можна побачити при застосуванні багатопроцесорного підходу. Важливо також зауважити, що вдалий вибір кількості робочих процесів залежить від кількості фізичних ядер процесора.

В якості рекомендацій щодо оптимізації обчислень на мові програмування Python можна запропонувати використання векторних обчислень, доцільно позбавлятись надлишкових операцій, уникати багаторазових операцій введення/виведення та використання глобальних змінних, а для виявлення місць в коді, що витрачають багато часу, використовувати профілювання, наприклад, за допомогою модуля cProfile.

Посилання

1. Delavallade, T., & Dang, T. H. (2007, July). Using entropy to impute missing data in a classification task. In *2007 IEEE International Fuzzy Systems Conference* (pp. 1-6). IEEE. https://doi.org/10.1109/FUZZY.2007.4295430

2. Initialization, Finalization, and Threads — Python 2.7.18 documentation. (б. д.). 3.12.3 Documentation. https://docs.python.org/2/c-api/init.html#threads

3. UCI Heart Disease Data. (б. д.). Kaggle: Your Machine Learning and Data Science Community. https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data

4. Zemlianyi, O., & Baibuz, O. (2024). Methods for imputing missing data on coronary heart disease. *System technologies*, *2*(151), 33-49. https://doi.org/10.34185/1562-9945-2-151-2024-04

5. concurrent.futures – Launching parallel tasks. (б. д.). Python documentation. https://docs.python.org/3/library/concurrent.futures.html